Lecture 12.2: GenAI: Diffusion Models

From Discrimination to Creation

Heman Shakeri

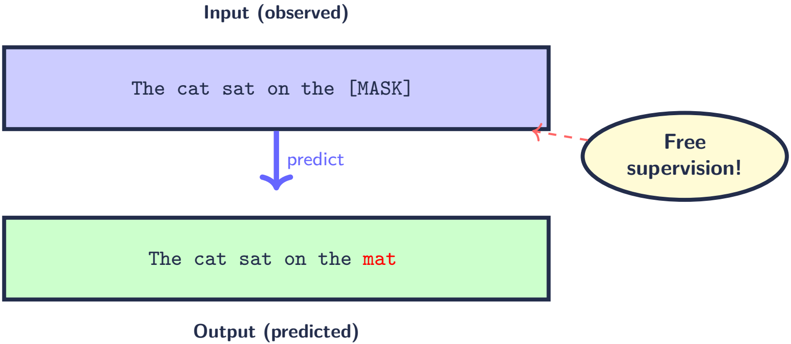
Self Supervised Learning

Most intelligence comes from unsupervised observation

- Babies learn how the world works largely by observation
  - Object permanence, gravity, intuitive physics
  - No explicit labels needed
- Humans learn to drive with $\sim$20 hours of practice
  - Leverage vast background knowledge from observation
  - Not millions of labeled examples
- **Common sense:** Generalized knowledge about the world
  - Taken for granted in humans
  - The "dark matter" of AI (LeCun & Misra, 2021)

Self-Supervised Learning: Recall the Core Idea

Learn to predict hidden parts from visible parts



Key insight: The data itself provides the training signal

- No manual labeling required
- Can scale to billions of examples

Self-Supervised Learning for Language

Two dominant strategies from Lectures 9–11:

Strategy	How it works	Examples
Masked Language Modeling	Mask 15% of tokens, predict them	BERT, RoBERTa
Autoregressive	Predict next token given all previous	GPT, LLaMA

Why this works for text:

- Discrete tokens: Can enumerate all possibilities
- Manageable length: Hundreds to thousands of tokens
- Natural ordering: Left-to-right for autoregressive
- Compactness: Can represent probability over entire vocabulary

We observed that these models learn rich semantic representations without labels!

Auto-regressive: One Step at a Time

Key Insight: We can re-frame the autoregressive for vision: Instead of regressing a blurry "average" pixel, we can predict a **probability distribution** over *discrete* pixel values (e.g., 0-255).

Process: (Factorization of the joint distribution)

1. Start with image missing all pixels
2. Predict first pixel **class**: $p(x_1)$ (Softmax over 256 values)
3. Predict second given first: $p(x_2 \mid x_1)$
4. Continue: $p(x_t \mid x_{<t})$ for all t

Factorization:

$$p(\mathbf{x}) = p(x_1) \cdot p(x_2 \mid x_1) \cdot p(x_3 \mid x_1, x_2) \cdots p(x_T \mid x_{<T})$$

Add diversity: Sample from the discrete distribution $x_t \sim p(x_t \mid x_{<t})$

Auto-regressive: Successes and Limitations

Modern successes:

- **GPT models** (including ChatGPT): Text generation, one token at a time
- **PixelCNN**: Image generation (Masked Convolution)
  - It uses Softmax over 256 pixel values!
  - Avoids blur by treating pixels as discrete classes, not continuous.
- **WaveNet**: Audio generation (Dilated Convolution)
  - Same trick: Uses Softmax over 256 *quantized* audio levels.

The Problem for Images:

Modality	Length	Auto-regressive?
Text (GPT)	Thousands of tokens	Efficient!
Images (512×512)	262,144 pixels	Too slow!

Challenge: The discrete approach works, but is computationally infeasible. Can we find a new way to handle continuous pixels **in parallel**?

The Vision Challenge: Why Not Just "Tokenize" Images?

Why can't we just apply text SSL (BERT/GPT) strategies to images?

e.g., Break image into 16x16 patches (like ViT) and treat them as "tokens"?

- **Problem 1: No Finite "Token" Vocabulary**
  - Text: We have a shared, discrete vocabulary ($\sim$50K tokens). We can use **Softmax**.
  - Images: A "patch" is a high-dimensional **continuous vector** (16x16x3 = 768 dims).
  - We cannot run Softmax over an infinite, continuous space!
- **Problem 2: The Averaging Problem Returns**
  - "Okay, so let's predict the continuous patch *vector* using **Regression** (MSE loss)."
  - **This fails!** The **target** (the patch) is a set of highly **correlated** pixels.
  - If a patch has multiple valid completions (e.g., pointy ear, floppy ear), the MSE loss forces the model to predict the **average vector**.
  - Result: A blurry, unrealistic patch. Predicting a correlated target fails!
- **Problem 3: No Natural Ordering**
  - Text has a clear 1D (left-to-right) structure for autoregression.
  - Images are 2D. A raster scan (row-by-row) of patches is arbitrary and inefficient.

The Key Insight: Change The Prediction Target

The Problem with Masking: The target (the masked patch) is a vector of **correlated** pixels. Predicting it with MSE causes the **averaging problem**.

The Solution (Diffusion): Change the task! Instead of predicting the *patch*, predict the *noise* we added.

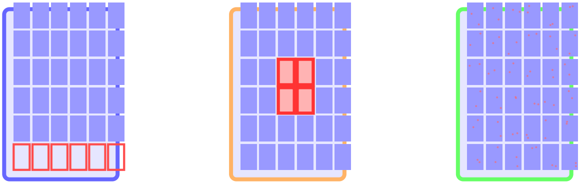
Why This Works (Denoising):

1. **New Target:** The target is now the **Gaussian noise vector ϵ**.
2. **Independence:** By definition, the noise ϵ is **uncorrelated** across all pixels.
3. **Averaging Solved:** We can now use a simple MSE loss ($\|\epsilon - \hat{\epsilon}\|^2$) to predict all 786,432 independent noise values **in parallel**!
4. **SSL Signal:** This is a perfect SSL task: we know the noise we added, so we have the ground truth for free.

Predicting independent noise avoids the averaging problem!

Visual Comparison: Three Strategies

Autoregressive | **Masking** | **Diffusion**



Autoregressive	Masking	Diffusion
1 pixel at a time 262K steps	Predict patch $\sim$1K steps	All pixels + noise 50-100 steps
Too slow!	Blurry!	**Optimal!**

Diffusion: $> 2000\times$ speedup over autoregressive for images!

The Diffusion Process: Overview

Two complementary processes:

1. **Forward (Fixed):** Gradually add noise over T steps
   - Start: Clean image $\mathbf{x}_0$
   - End: Pure Gaussian noise $\mathbf{x}_T \sim \mathcal{N}(0, \mathbf{I})$
   - *This is a fixed, known process*
2. **Reverse (Learned):** Gradually remove noise
   - Start: Pure noise $\mathbf{x}_T$
   - End: Clean image $\mathbf{x}_0$
   - *This is what we learn with a neural network*

If we know how to reverse the noising, we can generate!

Forward Process: Adding Noise

Iterative formulation:

$$\mathbf{x}_t = \sqrt{1-\beta_t} \cdot \mathbf{x}_{t-1} + \sqrt{\beta_t} \cdot \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})$$

where β_t is a **noise schedule**: $0 < \beta_1 < \beta_2 < \cdots < \beta_T < 1$

Key notation: Define $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$

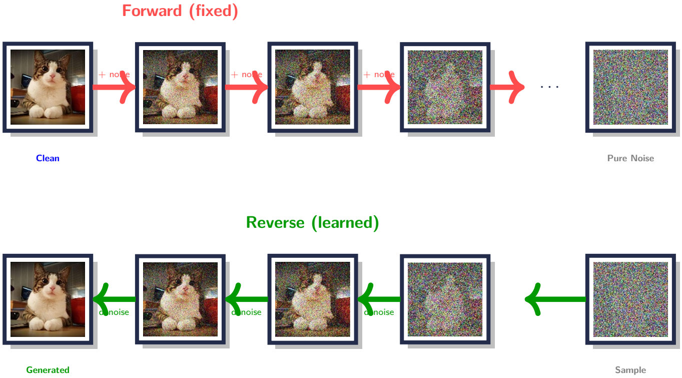
Closed-form: thanks to the properties of Gaussian distributions, we can analytically solve the entire sequential process (reparameterization trick):

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})$$

This closed form is **crucial** for efficient training!

Can jump directly to any timestep t without iterating

Diffusion Process Visualization



Challenge: How do we learn to reverse this process?

The Reverse Process: The Central Challenge

We have the (easy) Forward Process: We know how to add noise step-by-step: $p(\mathbf{x}_t \mid \mathbf{x}_{t-1})$

$$\mathbf{x}_0 \rightarrow \mathbf{x}_1 \rightarrow \cdots \rightarrow \mathbf{x}_T$$

We need the (hard) Reverse Process: To generate, we must learn to *remove* noise step-by-step: $p(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$

$$\mathbf{x}_0 \leftarrow \mathbf{x}_1 \leftarrow \cdots \leftarrow \mathbf{x}_T$$

The Problem: This reverse distribution $p(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ is **intractable**. It's unknown and depends on the entire (unknown) data distribution.

The Key Insight: It can be shown that this difficult reverse step becomes possible *if* we can estimate one thing: the gradient of the log-probability of the noisy data distribution, $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)$.

This gradient is the key to reversing the process. It has a name...

The Score Function

Definition: For probability distribution $p(\mathbf{x})$, the **score function** is:

$$s(\mathbf{x}) = \nabla_{\mathbf{x}} \log p(\mathbf{x})$$

This is the gradient of log-probability with respect to data

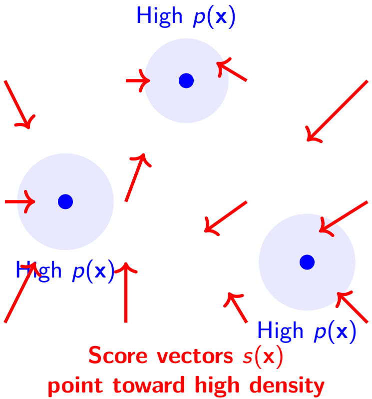
Intuition: Vector field pointing toward high-density regions

- At each point $\mathbf{x}$, $s(\mathbf{x})$ is a vector
- Points in direction where $\log p(\mathbf{x})$ increases most rapidly
- Following this field leads from noise to data!

For diffusion: We have score at each noise level t:

$$s_t(\mathbf{x}) = \nabla_{\mathbf{x}} \log p_t(\mathbf{x})$$

Score Function as Vector Field



Generation: Start from noise, follow the score to reach data

The Key Connection: Denoising is Score Matching

Tweedie's Formula: For noisy observation $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}$:

$$\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t) = -\frac{1}{\sqrt{1-\bar{\alpha}_t}}\mathbb{E}[\boldsymbol{\epsilon} \mid \mathbf{x}_t]$$

This means:

- The score function is proportional to expected noise
- Training to predict noise $\Leftrightarrow$ learning the score!
- so we don't have to learn the score function directly, instead we train a neural network $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$ to do self-supervised noise prediction ($\boldsymbol{\epsilon}$)!
- This is called **denoising score matching**

Training objective: Train $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$ to predict noise:

$$\mathcal{L} = \mathbb{E}_{t,\mathbf{x}_0,\boldsymbol{\epsilon}}\left[\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\|^2\right]$$

where $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}$

Why Predict Noise Instead of Images?

Alternative formulations:

- Predict clean image $\mathbf{x}_0$ directly
- Predict mean $\boldsymbol{\mu}_\theta$ directly

Why noise prediction is better:

1. **Simpler objective:** Just MSE loss on noise
2. **Better gradient flow:** Avoids predicting averages
3. **Connection to score:** $\epsilon_\theta \approx -\sqrt{1-\bar{\alpha}_t}\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)$
4. **Stationary target:** Noise is simple, stationary distribution

Once we predict noise, we can:

- Recover the score function
- Compute the denoising step to get $\mathbf{x}_{t-1}$

Training Procedure (Remarkably Simple)

For each training step:

1. Sample data point $\mathbf{x}_0$ from dataset
2. Sample random timestep $t \sim \text{Uniform}(1, T)$
3. Sample random noise $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})$
4. Create noisy version: $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\epsilon}$
5. Predict noise: $\hat{\boldsymbol{\epsilon}} = \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$
6. Minimize MSE: $\mathcal{L} = \|\boldsymbol{\epsilon} - \hat{\boldsymbol{\epsilon}}\|^2$

That's it! Just predict the noise you added

Sampling: Generating Images

To generate a new image:

1. Start with pure noise: $\mathbf{x}_T \sim \mathcal{N}(0, \mathbf{I})$
2. For $t = T, T-1, \ldots, 1$:
   - Predict noise: $\hat{\boldsymbol{\epsilon}} = \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$
   - Compute mean: $\boldsymbol{\mu} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\hat{\boldsymbol{\epsilon}}\right)$
   - Sample $\mathbf{z} \sim \mathcal{N}(0, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
   - Update: $\mathbf{x}_{t-1} = \boldsymbol{\mu} + \sigma_t \mathbf{z}$
3. Return $\mathbf{x}_0$

Intuition:

- At each step: predict and remove noise
- Add small random noise for stochasticity (except last step)
- Gradually reveal the image by following the score

From Self-Supervision to Creativity

A puzzle emerges:

- Diffusion models learn to denoise images (self-supervised)
- They're trained to predict noise as accurately as possible
- Yet they generate novel, creative images not in training data

The apparent contradiction:

1. Perfect learning $\rightarrow$ learns ideal score function exactly
2. Ideal score function $\rightarrow$ perfectly reverses forward process
3. Perfect reversal $\rightarrow$ only generates memorized training examples
4. But we observe: Creative, novel outputs!

The Central Puzzle

Question: How do diffusion models produce creative outputs?

Novel combinations not in training data?

The Paradox:

If model perfectly learns ideal score function on finite dataset, it can only memorize training data!

Why? For finite dataset $\mathcal{D} = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\}$:

$$p_t(\mathbf{x}) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{N}(\mathbf{x} \mid \sqrt{\bar{\alpha}_t}\mathbf{x}^{(i)}, (1 - \bar{\alpha}_t)\mathbf{I})$$

As $t \to 0$, posterior concentrates on nearest training image

Perfect training = only memorization

Again, where does creativity come from?

Creativity as Structured Failure

Theoretical Insight: Creativity arises because model *fails* to learn ideal score

Crucially: This failure is structured by inductive biases!

For CNN-based U-Net, two biases are key:

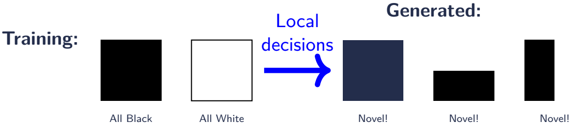
1. **Locality:** Finite receptive fields
   - Score at pixel (i, j) depends only on local neighborhood
   - Cannot coordinate globally instantaneously
   - Connects to self-supervision: Each patch makes independent denoising decisions
2. **Equivariance:** Weight sharing (Lecture 4!)
   - CNNs treat different locations similarly
   - Translation invariance
   - Connects to self-supervision: Denoising strategy learned on one patch applies everywhere

These architectural constraints prevent implementing an "Ideal Score Machine"

Result: The model denoises locally, composing globally novel mosaics.

The Simplest Example: Black and White Images

Training set: Only 2 images (all black, all white)



Exponentially many novel samples! (approximately 2^{N^2} for $N \times N$ image)

How? Each pixel independently decides its color based on local neighborhood

Local consistency: majority color in patch = center pixel

The Mechanism: Patch Mosaics

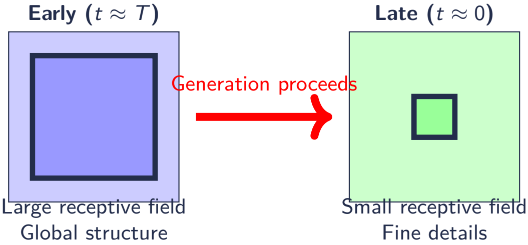
What happens instead of memorization:

1. **Local Bayesian Inference:**
   - Each pixel estimates local score using only nearby info
   - “Which training patch do I most resemble?”
2. **Mixing and Matching:**
   - Model doesn’t memorize whole images
   - Composes patches from different training images
3. **Locally Consistent, Globally Novel:**
   - Every small patch looks realistic (matches training)
   - Overall combination is new (never seen)
   - Combinatorial creativity!

Coarse-to-Fine Generation

Important empirical observation:

Effective receptive field shrinks during reverse process



Strategy:

- **Early:** Large patches set global structure (object type, layout)
- **Late:** Small patches add fine details (textures, edges)

Explaining Spatial Inconsistencies

Famous diffusion "errors":

- Hands with wrong number of fingers
- Clothing with incorrect number of arms
- Bifurcated shoes or multiple legs on pants

Mechanistic explanation: Excessive locality at late times ($t < 0.3$)

- Receptive field < 5 pixels
- Different parts of image cannot coordinate
- Each region independently decides "this should be a finger"
- Result: Too many fingers!

This is not a bug—it's a fundamental consequence
of the local score approximation that enables creativity

It's a trade-off!

Connection to Lecture 6: U-Net Returns!

Recall from Lecture 6: U-Net for image segmentation

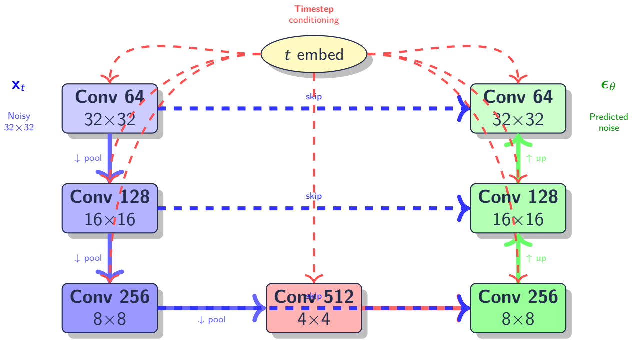
Perfect for diffusion because:

- **Spatial structure:** Preserves image layout
- **Multi-scale:** Handles coarse and fine details
- **Skip connections:** Essential for preserving details during denoising
- **Image-to-image:** Noisy image $\rightarrow$ noise prediction

Typical architecture: $\epsilon_\theta(\mathbf{x}_t, t)$

- **Input:** Noisy image $\mathbf{x}_t$ + timestep t
- **Output:** Predicted noise $\hat{\epsilon}$
- **Timestep embedding:** Sinusoidal encoding (like Transformers!)

U-Net for Diffusion



Skip connections are crucial: preserve high-frequency details lost in downsampling

The Challenge: Pixel-Space is Expensive

DDPM works, but:

- 512×512 image = 786,432 pixels
- Need 50-1000 denoising steps
- Running U-Net 1000 times on 512×512 is prohibitive!

Key Observation: Most image information is redundant!

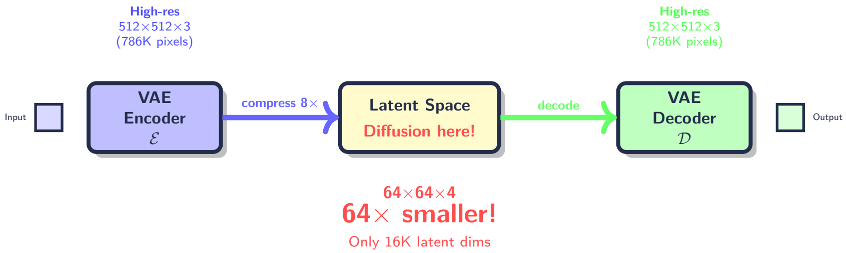
Nearby pixels are highly correlated

Solution: Latent Diffusion

Run diffusion in compressed latent space

Latent Diffusion Models (LDM)

Idea: Use pre-trained VAE to compress images



Process:

1. Encode image to latent: $\mathbf{z} = \mathcal{E}(\mathbf{x})$
2. Run diffusion on $\mathbf{z}$ (much smaller!)
3. Decode back: $\hat{\mathbf{x}} = \mathcal{D}(\mathbf{z})$

Benefits of Latent Diffusion

Why this is game-changing:

- **Speed:** 64×64 latent vs 512×512 pixels
  - $\frac{512^2}{64^2} = 64\times$ fewer pixels per step!
  - Same quality, dramatically faster
- **Memory:** Can train on consumer GPUs
  - 512×512 diffusion: needs A100 (80GB)
  - 64×64 latent: works on RTX 3090 (24GB)
- **Quality:** Still generate high-res images
  - VAE decoder upsamples from latent
  - Preserves details surprisingly well

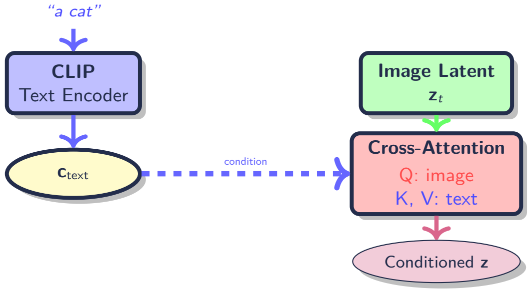
This is what Stable Diffusion uses!

Text Conditioning via Cross-Attention

Challenge: Generate specific content, not random images

Need to condition on text prompts!

Solution: Cross-attention between image and text



Mechanism: Image patches "query" text to find relevant semantic info

Cross-Attention Mechanism

Recall from Lecture 8: Attention mechanism

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}}\right)\mathbf{V}$$

For cross-attention in diffusion:

- $\mathbf{Q} = \mathbf{W}_Q \cdot \mathbf{z}_t$ (query from noisy image)
- $\mathbf{K} = \mathbf{W}_K \cdot \mathbf{c}_{\text{text}}$ (key from CLIP text)
- $\mathbf{V} = \mathbf{W}_V \cdot \mathbf{c}_{\text{text}}$ (value from CLIP text)

Intuition:

When generating cat's ear, image latent "attends to" "cat" in text embedding

Each image region focuses on relevant text concepts

Classifier-Free Guidance (CFG)

Problem: Text conditioning alone may be too weak

Solution: Amplify the conditioning effect!

Training: Randomly drop text 10% of time

- Model learns both $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, \mathbf{c})$ and $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, \emptyset)$

Sampling: Use guided prediction

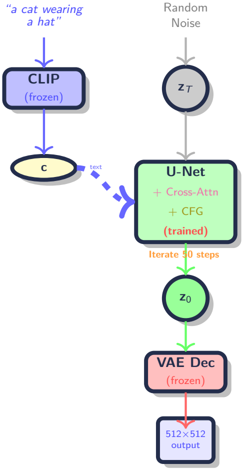
$$\tilde{\boldsymbol{\epsilon}} = \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, \emptyset) + s \cdot [\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, \mathbf{c}) - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, \emptyset)]$$

where $s > 1$ is guidance scale (typically 7.5)

Intuition: Move away from unconditional, toward conditional

Higher $s \to$ stronger conditioning but less diversity

Complete Stable Diffusion Pipeline



Components: CLIP (frozen) + U-Net (trained) + VAE (frozen)

Faster Sampling

Problem: DDPM needs 1000 steps, too slow!

Method	Steps	Description
DDIM	50-100	Deterministic, skip steps
DPM-Solver	20-50	ODE solver for diffusion
Flow Matching	10-20	Continuous flows
Consistency	1-4	Direct mapping

DDIM: Deterministic sampling with fewer steps

Make sampling deterministic, skip timesteps: 50 steps achieves similar quality

Diffusion Transformers (DiT)

Recent trend: Replace U-Net with Transformer blocks

DiT architecture:

1. **Patchify:** Split latent into patches (like ViT!)
2. **Add embeddings:** Position + timestep
3. **Transformer blocks:** Multi-head attention + FFN
4. **Unpatchify:** Reshape to latent space

Advantages over U-Net:

- Scalability: Easy to scale up
- Long-range dependencies: Global self-attention
- Unified architecture: Same as ViT, BERT, GPT

Results: DiT matches or exceeds U-Net quality with better scaling!

Evaluation Metrics

How do we measure quality?

Metric	What it measures	Range
FID	Distribution similarity	Lower better
CLIP Score	Text-image alignment	Higher better
Human Eval	Preference ratings	Subjective

FID (Fréchet Inception Distance):

- Compare feature distributions of real vs generated
- Extract features using Inception-v3
- Fit Gaussian, compute Fréchet distance
- Lower FID = closer to real data

What We've Learned: The Journey

1. **The Averaging Problem:**
   - Predicting multiple correlated values $\rightarrow$ blurring
   - Solution: Predict one at a time (auto-regressive)
2. **The Insight:** Break correlations with noise
   - Add independent Gaussian noise to all pixels
   - Can predict all simultaneously without averaging!
3. **Mathematical Foundation:**
   - Score function $s(\mathbf{x}) = \nabla_{\mathbf{x}} \log p(\mathbf{x})$ guides generation
   - Denoising = Score matching (Tweedie's formula)
   - Training: Just predict noise with MSE loss

What We've Learned: The Practice

4. Architecture: U-Net from Lecture 6

- Skip connections preserve details
- Multi-scale processing
- Timestep conditioning

5. Creativity Paradox:

- Perfect learning $\rightarrow$ memorization
- Creativity from structured failure
- CNN locality + equivariance $\rightarrow$ patch mosaics
- Coarse-to-fine: large patches (early) to small (late)

6. Stable Diffusion: Making it practical

- Latent diffusion ($64\times$ faster)
- Cross-attention for text conditioning
- Classifier-free guidance (amplify conditioning)

Key Takeaways

1. **Diffusion revolutionized image generation**
    - Stable training, high quality, excellent diversity
    - Solved problems that plagued GANs
2. **Self-supervised learning is key**
    - No labels needed, just images
    - Denoising provides natural training signal
3. **Architecture matters**
    - U-Net perfect for image-to-image tasks
    - Inductive biases shape creativity
    - Future: Transformer-based (DiT)
4. **Efficiency through clever design**
    - Latent space diffusion (64$\times$ speedup)
    - Cross-attention for conditioning
    - Classifier-free guidance for control

The Generative AI Revolution

From 2020 to 2025:

- 2020: DDPM introduces stable training
- 2021: DALL-E shows text-to-image, CLIP enables grounding
- 2022: Stable Diffusion democratizes (open source!)
- 2023: Midjourney reaches photorealism, video begins
- 2024: Sora generates minute-long videos
- 2025: Multimodal unified models

Applications:

- Art, design, advertising
- Scientific research (proteins, drugs)
- Text-to-video (Sora, Runway)
- Image editing, super-resolution

We're still in the early days!